Our Case No. 10519/75
(MD-99)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
APPLICATION FOR UNITED STATES LETTERS PATENT

INVENTORS: Christopher S. Moore
Matt Fruin
Colm Lysaght
Roy E. Scheuerlein

TITLE: Method for Making a Write-Once Memory Device Read Compatible with a Write-Many File System

ATTORNEY: Joseph F. Hetz
BRINKS HOFER GILSON & LIONE
P.O. BOX 10395
CHICAGO, ILLINOIS 60610
(312) 321-4719

Method for Making a Write-Once Memory Device Read Compatible with a Write-Many File System

Related Applications

This application claims the benefit of U.S. Provisional Application No. 60/282,723, filed April 9, 2001, and U.S. Provisional Application No. 60/282,790, filed April 9, 2001, both of which are hereby incorporated by reference.

Background

Non-volatile memory is becoming standard in consumer appliances such as digital cameras, digital audio players, and personal digital assistants. The demands for affordable non-volatile storage media coupled with the low cost per bit of information achievable in creating write-once storage elements have given rise to the proliferation of write-once memory devices in the consumer marketplace. Write-many file systems are typically not used to store data in write-once memory devices because those file systems are designed to re-write bookkeeping data structures stored in a given set of memory cells whenever data is added or deleted. Instead, specially-designed write-once file systems, such as ISO9660 and Universal Disk Format (UDF), are used. Unfortunately, data stored using a write-once file system typically cannot be read by a write-many file system. This interoperability problem can affect a large number of users since the prevalence of the MS-DOS FAT file system (ANSI/ISO/IEC 9293-1994) in the personal computer space has lead to the broad acceptance of this write-many file system in the nascent consumer appliance market as well.

There is a need, therefore, for a method that can be used to make a write-once memory device read compatible with a write-many file system and, more generally, to make write-once memory devices behave like write-many memory devices.

Summary

The present invention is defined by the following claims, and nothing in this section should be taken as a limitation on those claims.

By way of introduction, the preferred embodiments described below provide a method for making a write-once memory device read compatible with a write-many file system. In one preferred embodiment, a method for re-writing to a logical address of a write-once memory device is provided. A physical-to-logical address map is built from data stored in the memory device that associates individual physical addresses with individual logical addresses. When a logical address is re-written, data associating that logical address with a new physical address is stored, and data associating that logical address with an old physical address is invalidated. When the logical address is read, the physical-to-logical address map is used to read the new physical address instead of the old physical address. Other preferred embodiments are provided, and each of the preferred embodiments described herein can be used alone or in combination with one another.

The preferred embodiments will now be described with reference to the attached drawings.

Brief Description of the Drawings

Figure 1 is an illustration of a page of memory of a preferred embodiment in accordance with a model set forth in the SmartMedia™ specification.

Figure 2 is an illustration of a memory array of a preferred embodiment.

Figure 3 is an illustration of a physical-to-logical address map of the memory array of Figure 2.

Figure 4 is an illustration of the memory array of Figure 2 with a new FAT table stored therein.

Figure 5 is an illustration of a physical-to-logical address map of the memory array of Figure 4.

Detailed Description of the Presently Preferred Embodiments

By way of overview, the preferred embodiments described herein relate to methods that can be used to make a write-once memory device read compatible with a write-many file system. In one preferred embodiment, such compatibility is accomplished using a physical-to-logical address map. A "physical address" is the actual address of a memory location in a memory array, and a "logical address" is the address used by an application in a host device in referring to a memory location in the memory array. The physical and logical addresses may not be identical to one another, and a physical-to-logical address map provides a mechanism for translating a logical address into a physical address and vice versa. The physical-to-logical address map is built from data stored in the memory device that associates individual physical addresses with individual logical addresses. As described below, this data can take various forms, such as an entry in a table or in a field associated with a memory location. When a logical address is re-written, the data associating that logical address with a new physical address is stored, and the data associating that logical address with an old physical address is invalidated. When the logical address is read, the physical-to-logical address map is used to read the new physical address instead of the old physical address.

Memory devices and host devices operating in accordance with SmartMedia™ Physical Format Specifications Version 1.30, which is hereby incorporated by reference, are designed to create a physical-to-logical address map to replace defective blocks. A SmartMedia™ compliant memory device contains storage locations for data that associates individual physical addresses with individual logical addresses, and a SmartMedia™ compliant host device contains software that creates a physical-to-logical address map from this data. As mentioned above, the physical-to-logical address map is used by the SmartMedia™ compliant host device for block logical address to physical address conversions to replace blocks of memory found to be defective. In one preferred embodiment of this invention, software in a SmartMedia™ compliant host device is modified so that the physical-to-logical address mapping technique is also used to make a write-once memory device read compatible with a write-many file system. Before

turning to the modification, a brief description of physical-to-logical address mapping under the SmartMedia™ specification is provided.

A SmartMedia™ memory device is organized into blocks of memory, where each block comprises a plurality of pages of memory (*e.g.*, 16 or 32 pages). Figure 1 is an illustration of a page of memory in accordance with one model set forth in the SmartMedia™ specification. As shown in Figure 1, a page is organized into a storing area 10 and a redundant area (or zone) 20. The storing area 10 comprises two 256-byte data regions: Data Region 0 and Data Region 1. The redundant zone 20 is 16 bytes and comprises seven redundant fields: a reserved field (bytes 512-515), a data status flag field (byte 516), a block status flag field (byte 517), block address field-1 (bytes 518-519), ECC field-2 (bytes 520-522), block address field-2 (bytes 523-524), and ECC field-1 (bytes 525-527). Block address field-1 and block address field-2 are used to store the logical address of the block containing the page. In this way, block address field-1 and block address field-2 store data that associates a logical address to a physical address of the block.

If a block is found to be defective, an invalid flag is stored in the block status flag field of that block. (It should be noted that the "invalid flag" in this example does not necessarily indicate that data written to a page is incorrect.) This has the effect of invalidating the logical address stored in that block's block address field-1 and block address field-2. Data that is to be stored in a logical address associated with a defective block is stored instead in a new block, and the logical address is stored in the new block's block address field-1 and block address field-2. When software in the host device builds a physical-to-logical address map, the software reads the invalid flag stored in the block status flag field of the defective block and ignores the logical address stored in that block's block address field-1 and block address field-2. However, because an invalid flag is not stored in the block status flag field of the new block, the logical address stored in the new block's block address field-1 and block address field-2 is used to create an entry in the physical-to-logical address map correlating the logical address with the physical address of the new block.

The process described above for "marking a block bad" is performed by software in a SmartMedia™ compliant host device only when a block is defective. In one preferred embodiment, this software is modified so that a block is marked as bad when an attempt is made to re-write that block. This method allows a host device to re-write to a logical address even though the memory device is write-once. This technique can be used to re-write file system structures to make the write-once memory device read compatible with a write-many file system. A "file system structure" refers to any data that describes a partition in memory, the memory space within the partition, and/or the type of commands that a file system can or cannot use with that partition. Examples of file system structures include, but are not limited to, a file allocation table ("FAT" or "FAT table"), a root directory, and a sub-directory. An example of this preferred embodiment will now be provided with respect to storing a FAT table in a write-once memory device.

Figure 2 is an illustration showing the physical addresses, storing areas, and redundant zones of several blocks of a memory device. To simplify this illustration, of the seven fields in the redundant zone, only the block status flag field and block address field-1 are shown. As shown in Figure 2, a FAT table is stored in the storing area of a block at physical address 2, and logical address 1 is stored in that block's block address field-1. (In this example, a FAT table fits in a single block.) Additionally, logical address 0 is stored in block address field-1 of the block at physical address 1. Invalid flags are not stored in the block status flag fields of the blocks at physical address 1 and 2. Accordingly, when software in a host device creates a physical-to-logical map 40 of the memory device, it will create an entry in the map 40 correlating physical address 1 with logical address 0 and an entry correlating physical address 2 with logical address 1 (see Figure 3).

When data is added or deleted from the memory device, a new FAT table is written to the memory device. To ensure FAT file system compatibility, the new FAT must be stored at logical address 1, since logical address 1 is where the FAT file system expects the FAT table to be located in this example. When a command is received to store the new FAT table at logical address 1, the physical-to-logical map 40 is consulted to identify the physical address of the block (physical address 2). With reference to

Figure 4, the software in the host device then stores an invalid flag in the block status flag field of the block at physical address 2 to mark the block "bad," which effectively invalidates the data associating logical address 1 with physical address 2. The software in the SmartMedia™ compliant host device then stores the new FAT table in an available block (here, at physical address 4) and stores logical address 1 in that block's block address field-1. (It should be noted that the acts described above can be performed in a different order (*e.g.*, store the new FAT table then store the invalid flag).) As shown in Figure 5, when the physical-to-logical map 40 is updated, an entry is created correlating physical address 4 with logical address 1, and the entry correlating physical address 2 with logical address 1 no longer appears in the map 40. Accordingly, when a DOS FAT file system reads logical address 1, the new FAT table is read from physical address 4 instead of the old FAT table from physical address 2. As shown by this example, the logical address used by the DOS FAT file system remains the same, but the physical address associated with that logical address changes. By providing software re-mapping, the write-once memory device becomes read compatible with the DOS FAT file system.

If the host and memory devices use the ECC fields in the redundant zone, it is preferred that a new FAT table be written each time the FAT table is updated to ensure that the ECC stored in the redundant fields matches the data in the FAT table. If the ECC fields are not used, an existing FAT table can be updated by simply appending data to the existing FAT table. In that situation, a new FAT table is preferably written in a new block only if the update changed existing data in the FAT table. The same is true with other file system structures, such as root directories and sub-directories.

There are several alternatives that can be used. For example, while the block status flag field and block address field-1 were used in the above example, other fields in the redundant zone can be used instead of or in combination with these fields. For example, block address field-2 can be used instead of or along with block address field-1. Further, the data status flag field, which marks the data written to a page as invalid, can be used instead of the block status flag field, which marks a block as bad. Further, although fields in the redundant zone used in the above example were in compliance with SmartMedia™ specifications, redundant fields can be defined in a variety of ways not

compliant with the SmartMedia™ specifications. Accordingly, the term "redundant field" refers to one or more fields in a redundant area of a memory location, and a specific field in accordance with a specific specification should not be read into a claim containing this term unless that field is explicitly recited therein.

Additionally, while the DOS FAT file system was used in the above example, it should be noted that other write-many file systems can be used. Further, in addition to making a write-once memory device read compatible with a write-many file system, the method described above can be used with any data (not just file system structures) to emulate a write-many memory device using a write-once memory device.

The embodiment illustrated above used a legacy SmartMedia™ host device containing software that performed physical-to-logical address mapping when a block was marked bad due to a defect. The legacy SmartMedia™ host device was modified to allow the software to also mark a block as bad when an attempt was made to re-write a logical address. That embodiment allowed a write-once memory device to be compatible with a write-many file system with no modification to existing SmartMedia™ compliant memory devices. That embodiment may be especially preferred if a memory device does not have a hardware component that defines how the memory is written. However, some memory devices can include a hardware component, such as a controller, with a memory (*e.g.*, RAM) to create a physical-to-logical address map. The controller would automatically convert addresses sent from a host device to the proper physical location in the memory based on the map. The controller can sit in front of a memory array in a memory card such as a Multi-Media Card, Secure Digital Card, or CompactFlash Card. By performing the re-mapping in hardware in the memory device, the host device would be blind to the re-mapping operation. In another alternative, a hardware component (instead of a software component) can be used in the host device to perform the re-mapping. More generally, hardware and/or software in the host device or the memory device can be used to create a physical-to-logical address map and to store and invalidate data in the memory device that associates individual physical addresses with individual logical addresses. This functionality can also be distributed between hardware and/or software components in the host device and/or memory device.

In the preferred embodiments described above, the data associating an individual physical address with an individual logical address was stored in a redundant field of a memory location. In an alternate embodiment, this data is stored in a separate table in the memory device. Analogous to a FAT table, as changes are made, data in this table can be updated to point to new areas. This table can be organized as a physical-to-logical table or a logical-to-physical table. Invalidating an entry in the table as bad can be done by any suitable method. For example, the entry can be "obliterated" overwriting at least a portion of the stored data with a destructive pattern, as described in U.S. Patent Application Serial No. 09/638,439, which is assigned to the assignee of the present invention and is hereby incorporated by reference. Additionally, a temporal-to-spatial mapping technique can be used to find the most-recent entry to the table. In this way, the act of storing new data invalidates the old data. The temporal-to-spatial mapping technique is described in U.S. Patent Application Serial No. 09/748,589, which is also assigned to the assignee of the present application and hereby incorporated by reference. Additionally, an invalid flag can be stored for the data associating a physical address with a logical address to invalidate that data. As used herein, the term "invalid flag" refers to one or more bits that are recognized as invalidating data that is associated with the flag.

In another alternate embodiment, mapping logical addresses to physical addresses is performed primarily or entirely by circuitry in the memory device. The logical-to-physical redirection is accomplished in the memory device transparent to the user. A mapping table is provided in bit locations beyond the bit location of the user's bits in a block of data. In the write once memory of this preferred embodiment, the initial state is called Logic 1. Therefore, the initial entry of the map data is all 1's. This initial state is interpreted by the logic to mean the logical address is not redirected but that the logical address equals the physical address. Every block of data has a table entry. Preferably, the size of the data block is the size of the redundant replacement unit. The table entry can be changed to the address of any redundant block physical address on the chip. It is also possible to use associations of logical to physical addresses that restrict the choice to less than the entire chip. However, due to the one-to-one correspondence of table entries to logical addresses, the full choice of chip locations is efficient.

When an entry other than 1's is found in the table, the logical address is not used, but the chip logic uses the stored mapping data in the table as the block physical address. The block physical address is the location of a redundant block of memory locations. Preferably the redundant block address is different from any valid block logical address on the memory device. The address redirection is preferably combined with the operation on the memory device of moving the block of data from a temporary storage location called a page register to or from the main memory array locations.

An example of the logic flow that could be used to carry out the redirection is in two parts, *i.e.*, a read operation and a write operation. The table entry could have a "used" indicator bit and a redundancy physical address.

Read Command: Read the table entry for address at the logical page address. If all 1's, read data at the logical address. If not all 1's, read the table data at the redundant physical address. Repeat until all 1's and the user data can be read.

Write command: Read the table entry. (a) If all 1's, including the used bit, write data at the logical address, then set the used bit to 0. Then (b) If the table entry has an address other than 1's, read the table entry at the indicated physical address location and start at the top of this decision tree again. Then (c) if the used bit is set to 0, choose a redundant location that has not been used, write data at the chosen location, write the used bit at the chosen location, and write a table entry at the original table entry. A table address entry indicates that the original data is no longer valid.

Partial write operations are accomplished by an enhancement to decision step (c). Partial write is a request to write a block of data while the user has only given valid data bits for a portion of the block. Other data bits are assumed to be 1 or unchanged. The enhancement to (c) is a step (d) If the table address is all ones, try to write the data at the address and then read to verify the write. If successful, write the used bit to 0. If not successful, choose a redundant location that has not been used, write data at the chosen location, write the used bit at the chosen location, and write a table entry at the original table entry.

Alternatively, the used bit is unnecessary if every write is done with write verify logic. (a) is a write verify operation (d), and (b) is the same, (c) is not needed.

Advantages include that the user need not build a mapping table and neither block flags nor block address fields modifications are required. Increased on-chip logic and a small number of extra memory cells are needed and additional latency to read multiply rewritten data is incurred. This could be the preferred solution for some applications.

As described above, the memory device in these preferred embodiments is a write-once memory device. A write-once memory device comprises field-programmable storage locations that are fabricated in an initial, un-programmed digital state and can be switched to an alternative, programmed digital state at a time after fabrication. For example, the original, un-programmed digital state can be identified as the Logic 1 (or Logic 0) state, and the programmed digital state can be identified as the Logic 0 (or Logic 1) state. Because the memory cells are write-once, an original, un-programmed digital state of a storage location (*e.g.*, the Logic 1 state) cannot be restored once switched to a programmed digital state (*e.g.*, the Logic 0 state). A memory device is considered write-once even if it contains a secondary write-many memory unit. The preferred embodiments described herein can also be used with a write-many memory device.

The memory device can take any suitable form, such as a solid-state memory device (*i.e.*, a memory device that responds to electrical read and write signals to cause digital information to be read from and stored in a memory array of the device), a magnetic storage device (such as a hard drive), or an optical storage device (such as a CD or DVD). A memory device can comprise memory cells organized in a two-dimensional or three-dimensional array. In one preferred embodiment, the memory device takes the form of a solid-state memory device having a three-dimensional array of non-volatile write-once memory cells, as described in U.S. Patent No. 6,034,882 to Johnson et al., U.S. Patent No. 5,835,396 to Zhang, and U.S. patent application serial no. 09/560,626, all of which are hereby incorporated by reference. As discussed in those documents, three-dimensional memory arrays provide important economies in terms of reduced size and associated reductions in manufacturing cost. Although any suitable type of memory cell can be used, in one preferred embodiment, the memory cell comprises an anti-fuse and a diode. The memory array can be made from any suitable material. In one preferred embodiment, the memory array comprises a semiconductor material. Other materials can

be used, such as, but not limited to, phase-change materials and amorphous solids as well as those used with MRAM and organic passive element arrays, as described in U.S. Patent No. 6,055,180, which is hereby incorporated by reference. It is important to note that the following claims should not be read as requiring a specific type of memory array (*e.g.*, two-dimensional or three-dimensional) or material unless explicitly recited therein.

In one preferred embodiment, the memory device takes the form of a modular, compact, handheld unit, such as a memory card or stick, that comprises an external electrical connector that can be coupled with a host device. As used herein, the term "coupled with" means directly coupled with or indirectly coupled with through one or more intervening components. The host device can be a data storage device and/or a data reading device and can take the form of a digital camera, digital audio player, or other portable consumer product, for example. The memory device and host device can take other forms.

It is intended that the foregoing detailed description be understood as an illustration of selected forms that the invention can take and not as a definition of the invention. It should also be noted that a host device or memory device in compliance with the SmartMedia™ specification is not required by the claims unless explicitly recited therein. It is only the following claims, including all equivalents, that are intended to define the scope of this invention. Finally, it should be noted that any aspect of any of the preferred embodiments described herein can be used alone or in combination with one another.